# METHOD FOR ECU CALIBRATION AND DIAGNOSTICS DEVELOPMENT

## CROSS-REFERENCE TO RELATED APPLICATIONS

**[0001]** This application is a continuation-in-part of United States Patent Application No. 10/366,167 filed on February 13, 2003. The disclosure of the above application is incorporated herein by reference.

## FIELD OF THE INVENTION

**[0002]** The present invention relates generally to an electronic control unit (ECU) calibration and diagnostics development and, more particularly, to a technique that enables calibration tools to overwrite user selected random access memory (RAM) variables without any modifications or access to the underlying ECU source code.

## BACKGROUND OF THE INVENTION

**[0003]** In modern automotive electronic control units, there exist a number of embedded control algorithms that control different aspects of the vehicle. For instance, there may be an algorithm that controls the amount of fuel injected into the cylinders and a different control algorithm that is responsible for shifting gears in the transmission. These algorithms are developed in a way that they can be used on many different vehicle types, engines, and transmission for a variety of markets with different emission and safety requirements. During real-time execution, each of these algorithms use what is termed "calibrations" or "parameters" to adapt itself to the vehicle and requirements that it is controlling.

**[0004]** In the process of adjusting or "calibrating" an automotive ECU, often it is required to obtain a given set of operating conditions. Exemplary scenarios may include: (1) in order to calibrate or test an algorithm that controls the engine cooling fan(s), the engine coolant temperature must be greater than a given temperature; (2) in order to calibrate or tests an algorithm that controls the shifting of an automatic transmission from second to third gears, it is required that the vehicle speed be high enough to trigger the shift; (3) in order to calibrate or test an algorithm that adjusts the amount of fuel going to the cylinders of the engine based on altitude, it is often necessary to drive the vehicle across a range of altitudes from sea level to high mountain territory; and (4) when the catalytic converter temperature exceeds a safe value, one or more algorithms may be activated in an attempt to cool down the converter before damage occurs. Obtaining the necessary operating conditions can often be complex, costly and difficult.

**[0005]** However, the present invention provides a technique for overwriting user selected random access memory (RAM) variables. For instance, the RAM variable associated with the measured coolant temperature may be set to a value that will turn on the algorithm that controls the cooling fan(s). In another instance, the RAM variable associated with the vehicle speed may be set to a value higher than the 2-3 shift trigger point to force the transmission into a desired gear. In yet another instance, the RAM variable associated with the vehicle altitude calculation may be set to a value that simulates the desired operating altitude. In this way, various operating conditions

may be simulated without any modifications or access to the underlying ECU source code.

**[0006]** Similarly, the present invention may be employed to achieve actual operating conditions that may be difficult to otherwise obtain. For example, sometimes it is difficult in a vehicle equipped with an automatic transmission to achieve a condition where the engine rpm is very low while the engine torque is high because the transmission normally will downshift to a lower gear. With the present invention it is possible to overwrite the commanded gearstate of the transmission and force the transmission to any desired state regardless of engine rpm and torque load. In another example, it is sometimes necessary to quickly degrade/wear clutches inside an transmission or torque converter to simulate a high mileage operating condition. One way of degrading clutches is to allow slippage for an extended period of time between the friction material of these clutches. With the present invention it is possible to force a clutch into a slippage condition by overwriting and controlling the RAM variable associated with the desired clutch state (unlock, partial lock, full lock). Thus, it is desirable to provide a technique for simulating an embedded controller and the algorithms in the embedded controller without the need to provide physical stimulation to the embedded controller.

## SUMMARY OF THE INVENTION

**[0007]** In accordance with the present invention, a method is provided for controlling one or more RAM variables in a microprocessor without modifications to the underlying source code. The method includes: presenting an software program having a plurality of machine instructions of a finite number of fixed lengths in an executable form; searching through the machine instructions of the executable and finding at least one appropriate instruction to replace; defining a replacement instruction for identified machine instructions in the software program; and replacing identified machine instructions in the executable form of the software program with the replacement instruction. In one aspect of the present invention, the replacement instruction is further defined as a branch instruction that references an address outside an address space for the software program.

**[0008]** For a more complete understanding of the invention, its objects and advantages, reference may be had to the following specification and to the accompanying drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0009]** Figure 1 is a flowchart illustrating a method for controlling RAM variables in a microprocessor without modifications to the underlying source code in accordance with the present invention;

**[0010]** Figure 2A is a diagram illustrating an unmodified program memory image for a target software program embedded in a microprocessor;

**[0011]** Figure 2B is a diagram illustrating a program memory image modified in accordance with the present invention;

**[0012]** Figures 3A and 3B are flowcharts illustrating exemplary embodiments of relocation code in accordance with the present invention;

**[0013]** Figure 4 is a diagram depicting an exemplary embodiment of a calibration tool that is configured to support the present invention; and

**[0014]** Figure 5 is a diagram depicting exemplary logic to determine relocation instructions.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

**[0015]** In accordance with the present invention, a method is provided for controlling one or more RAM variables in an executable software program embedded in a microprocessor without modifications to the underlying source code. While the following description is provided with reference to RAM variables, it is readily understood that this technique is also applicable to manipulating any portion of the random access memory space associated with the software program.

**[0016]** Referring to Figure 1, the target software program is provided at step 30 in an executable form that is defined by a plurality of machine instructions of a finite quantity of fixed lengths. In many conventional microprocessors, machine instructions are limited to a finite quantity of fixed lengths. For example, machine instructions in a PowerPC-based processor are 32 bits in length; whereas Tri-Core and ST10 processors have machine

instructions that are 16 bits and 32 bits in length. Thus, some machine instructions in the underlying software may be replaced with other machine instructions. Although this concept serves as the basis for the present invention, it is readily understood that the broader aspects of the present invention may be extended to microprocessors having machine instructions of a varied length.

**[0017]** To manipulate a given RAM variable, one or more machine instructions having access to the variable are replaced with replacement instructions. Machine instructions to be replaced are first identified at step 32. These machine instructions are referred to herein as relocated instructions. Identification of a machine instruction includes determining a location or address for the machine instruction inside the software executable as further described below.

**[0018]** In the preferred embodiment, the relocated instructions are preferably machine instructions that access or modify variables that correspond to the RAM variables that are desired to be controlled. For instance, since there are no machine instructions that directly modify the memory space of a PowerPC-based processor, the code must load the value of the variable into a register, modify the value (e.g., though a math operation), and then store the modified value back into its appropriate memory space. Thus, in a PowerPC-based processor, the relocated instructions are specifically designated as load and/or store instructions for the applicable variables in the target software. However, it is readily understood that other types of machine instructions may also serve as relocated instructions.

**[0019]** Next, an appropriate replacement instruction is defined at step 34 for each of the identified machine instruction. Each replacement instruction is preferably defined as a branch instruction that references an address outside the memory space for the target software program. In the preferred embodiment, branch instructions pass processing control to a series of machine instructions that are defined in the unused portion of the memory space and are referred to herein as relocation code. Relocation code is responsible for obtaining the desired value for the given RAM variable and writing this value into the given RAM variable as further described below.

**[0020]** Alternatively, it is envisioned that replacement instructions may be defined as instructions that cause an interrupt or an exception to occur in the microprocessor. For example, PowerPC-based processors provide a "sc" command to perform this function; whereas ST10-based processors provide a "trap" instruction to perform this function. Although these types of instructions provide an alternative technique for branching to a different address space, this approach is generally not preferred because these types of instructions may interfere with the normal operation of the microprocessor.

**[0021]** In another alternative, replacement instructions are defined as no operation instructions that cause the microprocessor to perform no function. Replacing instructions that access a given RAM variable with a no operation instruction prevents the microprocessor from updating the variable. Thus, it is envisioned that a desired value may be placed into the RAM variable using available standard features of the ECU. For example, ECU controller available

from Ford Motor Company includes a POKE feature, whereby a calibration tool can write a value to RAM location. The present invention extends this preexisting functionality by disabling the ECU from subsequently writing to given RAM addresses, thereby enabling the calibration system complete control over the value of the given RAM addresses. It is understood that there are other methods for writing a value into a RAM location; examples include: a dual ported RAM memory, BDM (Background Debug Mode) port on PowerPC-based microcontrollers and an AUD (Advanced User Debug) port on microprocessors from Hitachi.

**[0022]** The identified machine instructions of the target software program are then replaced at step 36 with replacement instructions. Specifically, replacement instruction are inserted into the program memory image of the software program at the identified address. Lastly, the relocation code is also inserted at step 38 at a location outside the software executable program space.

**[0023]** Figure 2A illustrates an unmodified program memory image 40 for a target software program embedded in a microprocessor. The memory space may be partitioned into an address space 42 for the target software program and an unused portion 44 of memory space. It is readily understood that the address space 42 for the target software program may be further partitioned into a data portion and a code portion.

**[0024]** Figure 2B illustrates a program memory image modified in accordance with the present invention. One or more relocated instructions designated at 46 may be replaced with replacement instructions. Replacement

instructions in turn pass processing control to relocation code 48 that is defined in a memory space outside of the memory space for the target software program.

**[0025]** Figures 3A and 3B are flowcharts that illustrate exemplary embodiments of relocation code in accordance with the present invention. In general, the relocation code performs four primary functions. Referring to Figure 3A, the relocation code initially determines if the relocation feature is enabled or disabled as shown at step 52. When the relocation feature is disabled, the relocation code executes the relocated instruction as shown at step 56; and then processing branches at step 66 to the machine instruction following the relocated instruction in the target software program.

**[0026]** On the other hand, when the relocation feature is enabled, the relocation code performs the following functions. The relocation code reads the desired value of the RAM variable at step 60 from a location which may be modified by the calibration system. The relocation code then stores the desired value at step 62 into the original RAM variable.

**[0027]** Lastly, the relocation code branches processing at step 66 to the machine instruction following the relocated instruction in the target software program. The relocation code described above assumes a direct addressing method of storing variable values. In other words, each machine instruction that manipulates the value of a variable contains the address information needed to access that variable in memory.

**[0028]** In some instances, an indirect addressing method may be employed for loading or storing variable values. Indirect addressing first loads

the address of a variable into a machine register, and then uses the register to load or store the value of the variable. Thus, it is not possible to directly determine what store or load instructions are associated with a given variable. For these types of instructions, the present invention determines the machine register used as the pointer to the variable and then searches, starting from the specified instruction, for all store instruction using that register. The search includes all instructions in the current function (or routine) as well as all function that may be called by the function. In this way, all instructions that have the possibility of being load or store instructions to the given variable are modified. With this method, it is possible to modify a load or store instruction that is not associated with the given variable.

**[0029]** Relocation code for an indirect addressing method of storing variable values is shown in Figure 3B. In this case, it is necessary for the relocation code to determine that the value of the register is in fact pointing to the given variable as shown at step 55; otherwise the relocation code is in a manner as set forth in relation to Figure 3A.

**[0030]** Conventional calibration tools may be configured to support the present invention as shown in Figure 4. Calibration tools are generally configured to calibrate and test software-implemented control algorithms which may be embedded in an automotive electronic control unit (ECU). Exemplary calibration tools are commercially available from Accurate Technologies Inc., dSPACE GmbH, ETAS GmbH, and Vector GmbH. While the following description is provided with reference to control algorithms embedded in an

automotive electronic control unit, it is readily understood that the broader aspects of the present invention are applicable to other types of software applications which are embedded in microprocessors.

**[0031]** The calibration tool 70 is generally comprised of a user interface 72, an ECU interface 74 and, optionally a memory emulator 80. A user configures the calibration environment through the use of the user interface 72. In general, the user interface may be used to specify the target software program and the RAM variables that are desired to be controlled. In addition, the user may further specify the machine instructions which are to be relocated as well as the corresponding replacement instructions. One skilled in the art will readily recognize that a suitable user interface may be designed to support these required functions of the present invention.

**[0032]** In accordance with the present invention, the calibration tool 70 may be further configured to include an instruction locator 76 and an instruction replacement component 78. The instruction locator 76 is adapted to receive a specified RAM variable address, and data type within a target software program and operable to identify location information for the machine instructions associated with reads or writes to the RAM address within the executable form of the target software program. In one exemplary embodiment, the instruction locator 76 searches through the application image (hex record) for the target software and parses each machine instruction therein. For Embedded Application Binary Interface (EABI) compliant compilers, load and store instructions can be identified in the application image. In PowerPC-based

processors, running software that is EABI compliant, registers must be used in specific ways. For example, R13 must be used to point to the small data area for read/write memory. This register normally is used to index to the internal random access memory of the processor. Using this specific information, the instruction locator has the ability to reverse calculate an address for any load or store instruction in the small data area.

**[0033]** Another example for obtaining address information from an instruction is using machine register R2. This register normally contains a pointer to read only memory in the address space of the target processor. Sometimes pointers or address values are stored in the read only memory to store the location of certain RAM variables or ROM parameters. It is possible to interpret a load or read instruction that uses the R2 register and determine what address is being read or loaded. In this way, the instruction locator as defined later, can also read the read only address containing address or pointer information. It is readily understood that other techniques for identifying location information for a specific machine instruction are within the broader aspects of the present invention.

**[0034]** The instruction replacement component 78 is then operable to replace the specified machine instruction with a replacement instruction. To do so, the instruction replace component 78 is adapted to receive the replacement instruction and then insert the replacement instruction into a program memory image of the software program at the identified address. In the exemplary embodiment, the instruction replacement component 78 also generates the

applicable relocation code and inserts the relocation code into an unused portion of the memory space on the target microprocessor.

**[0035]** The modified program memory image is then FLASHED or downloaded into the ECU for execution. During execution, the replacement instructions are executed in place of the original machine instructions which accessed a given RAM variable. In this way, the RAM variable is being controlled by the calibration tool in a manner different than was originally contemplated by the ECU. In one exemplary embodiment, the calibration tool includes an ECU interface 74 which is operable to download the modified program memory image into the ECU. However, it is readily understood that other techniques may be employed to FLASH the ECU. Moreover, it is to be understood that only the relevant steps of the process are discussed herein, but that other software-implemented features may be needed to manage and control the overall calibration tool.

**[0036]** In order to access constants within a control algorithm embedded in a read-only memory space, it is envisioned that the calibration tool may also employ a memory emulator 80 or other interface to the ECU, such as CCP (Can Calibration Protocol). A suitable memory emulator is the M5 memory emulator which is commercially available from the Accurate Technologies, Inc.

**[0037]** An exemplary algorithm employed by the instruction locator is further described in relation to Figure 5. Briefly, the algorithm searches for machine instructions that contain addressing information related to the given RAM variables. To begin, a first machine instruction residing in the program

executable is read at step 82. For each machine instruction, a determination is made at step 84 as to whether the instruction contains address information related to the desired RAM variable. Some examples of instructions that may contain addressing information for a PowerPC microcontroller are the addi, ori, stfs, and ldfs instructions. However, it is readily understood that other instructions which contain address information are within the scope of the present invention. In addition, one readily skilled in the art will appreciate that more than one machine instruction may be used to determine addressing information for the desired RAM variable. For example, in a PowerPC-based microcontroller an addis instruction followed by an ori instruction can be used to load an address into a machine register.

**[0038]** When the current instruction contains address information for the desired RAM variable, a determination is then made at step 86 as to whether the current instruction is a load or a store instruction. If the current instruction contains address information for the desired RAM variable and is a load or a store instruction, then the instruction is saved in a list as a candidate instruction for replacement as shown at step 88. On the other hand, when the current instruction does not contain information for the desired RAM variable, processing continues with the next machine instruction as shown at step 94.

**[0039]** When the current instruction contains address information for the desired RAM variable, but is not a load or a store instruction, then a determination is made at step 90 as to whether the instruction loads an address for the RAM variable into a processor register. If the instruction does load the

address, then processing proceeds as follows. Every machine instruction contained in the current function and all possible called functions following the instruction that load the address into the machine register are read and evaluated at step 92. Each instruction identified as a load or a store instruction that uses the given register is again saved in the list of candidate instructions. Processing then continues with the next machine instruction at step 94 until the last instruction of the program is encountered.

**[0040]** While the invention has been described in its presently preferred form, it will be understood that the invention is capable of modification without departing from the spirit of the invention as set forth in the appended claims.